计算机组成

虚拟存储器

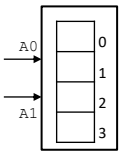
高小鹏　　杨建磊

北京航空航天大学计算机学院

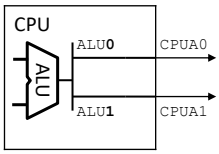
目录

CPU地址与存储器地址是什么关系？

- 存储器参数：容量为4B；2位地址信号，即A[1:0]
  - 存储器内部4个单元的地址编号分别为00b，01b，10b，11b

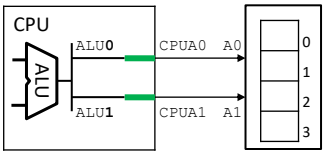


- CPU地址参数：2位地址信号，即ALU[1:0]
  - 对于load/store类指令，cpu地址是由ALU计算产生的
    - 暂时不考虑PC产生的指令地址
  - 为简化分析，假设ALU的计算结果位宽为2位，即ALU[1:0]



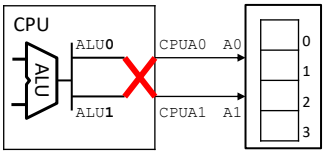
CPU地址与存储器地址是什么关系？

- 我们很容易认为cpu地址与存储器地址是完全相同的
  - 例如，如果cpu产生的地址为00b，则必然对应存储器的00b地址
- 事实上，只有当ALU的ALU[i]与存储器的A[i]连接才会是这样



A.常规连接方式

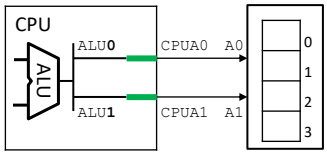
- 如果改变两者的地址信号的连接关系，会发生什么呢？



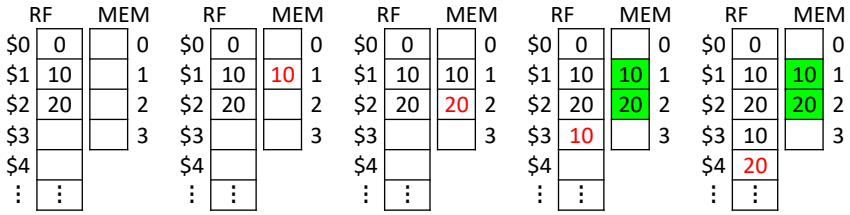
B. 非常规连接方式

地址信号连接方式对程序执行结果的影响

- 假设$1为10，$2为20
- 常规连接方式执行结果

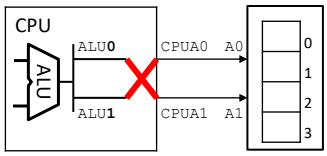



```
1：sb $1, 1($0)
2：sb $2, 2($0)
3：lb $3, 1($0)
4：lb $4, 2($0)
```

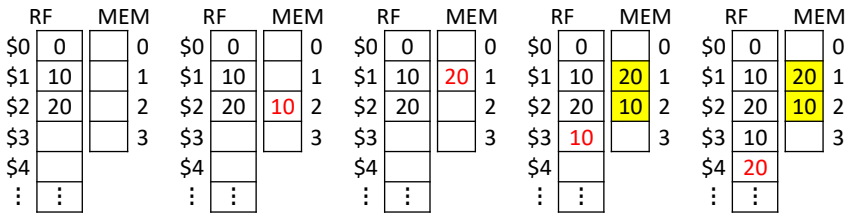


地址信号连接方式对程序执行结果的影响

- 假设$1为10，$2为20
- 非常规连接方式执行结果

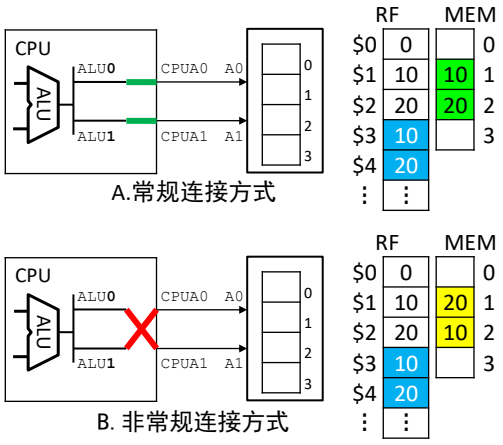



```
1：sb $1, 1($0)
2：sb $2, 2($0)
3：lb $3, 1($0)
4：lb $4, 2($0)
```



地址信号连接方式对程序执行的影响

- 存储类器件的区别
  - 寄存器：没有任何变化
  - 主存：存储的数据发生了交换



A.常规连接方式

B. 非常规连接方式

- 程序运行存在正确性问题吗？不存在正确性！
  - load/store类指令对某个cpu地址的读写，始终是由特定的存储单元承载
  - 只要对应关系在确定后不随意更改，就不影响程序正确性

地址信号连接方式对程序执行的影响

- 程序是否存在正确性，要看从什么角度出发
  - 程序是程序员编写的。只要程序员“认为”正确，那就是正确！
- 从存储器的角度
  - 在不同的映射方式中，10和20对应的存储器单元是不同的
- 从程序员的角度
  - 无论采用何种地址映射方式，他始终看到的“地址1存储的是10，地址2存储的是20”

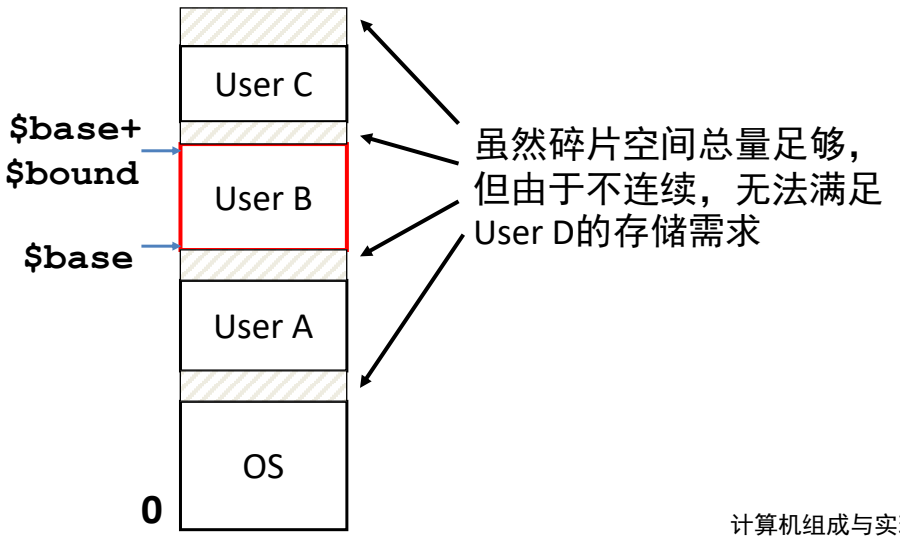
程序员看到的地址，本质上是ALU计算出的结果

ALU的计算结果再通过信号连接，就实现了cpu地址与存储器地址的映射

目录

早期的解决方案

- 针对每个程序的存储需求，给每个程序设置一个基地址和边界
- 虽然OS可以管理多个程序了，但无法解决碎片问题

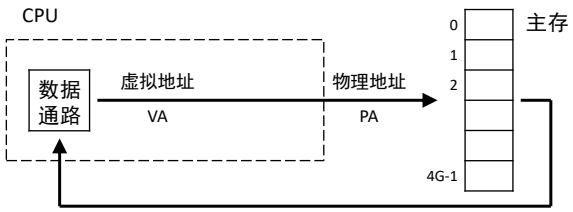


虚地址与物理地址

- 由ALU产生的地址计算结果被称为虚拟地址(VA，Virtual Address)，由全部虚拟地址构成的集合被称为虚拟地址空间
  - 之所以用“虚拟”，是因为这些地址是计算产生的
- 存储器单元的地址被称为物理地址(PA，Physical Address)，由全部物理地址构成的集合被称为物理地址空间
  - 之所以用“物理”，是因为这些存储单元是客观存在的

使用物理地址的系统

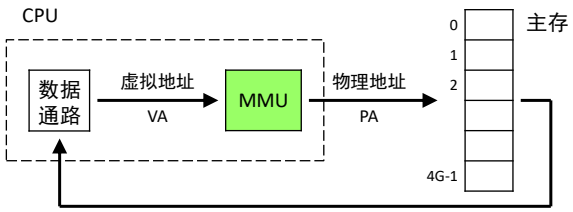
- 地址特点： VA与PA是完全相同的
- 实现方式： VA与PA是直接连接在一起的，



- 应用场景：简单的嵌入式系统
  - 例如，汽车abs控制系统、电梯控制器等的嵌入式微控制器

使用虚拟地址的系统

- 地址特点： VA与PA是不同的
- 实现方式： VA通过MMU转换为PA

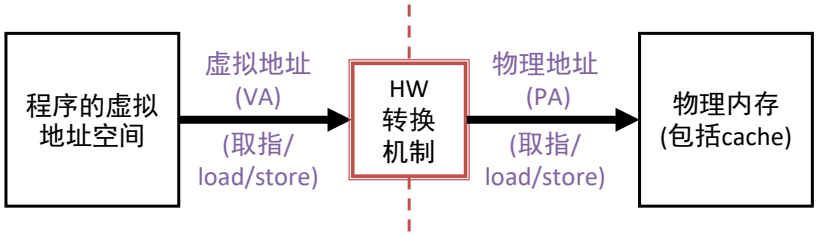


- 应用场景：现代计算机系统，运行复杂的多任务操作系统
  - 例如：服务器、台式机、笔记本、智能手机等

目录

虚拟地址到物理地址的转换

- 每个程序只使用它自己的虚拟地址，并且“认为”它是唯一运行的程序
- 程序之间存在保护机制
- OS能决定哪个程序运行在主存的哪个区域
- 硬件实现虚拟地址到物理地址的转换



计算机组成与实现

分页

- 把虚存与主存都分为固定大小的若干块，每块称为页面（page）
  - 典型的页面容量为4KB或8KB
  - 由于页面大小固定，因此将地址相应的低位部分称为页内偏移(page offset)
  - 虚存的每个页面被称为虚页面(virtural page，VP)
  - 主存的每个页面被称为物理页面(physical page，PP)
- VA地址分割为

virtural page #	offset

  - 虚页号(virtural page number，VPN)，页内偏移(page offset)
    - VPN：VA去除offset后的高位部分
- PA地址分割为

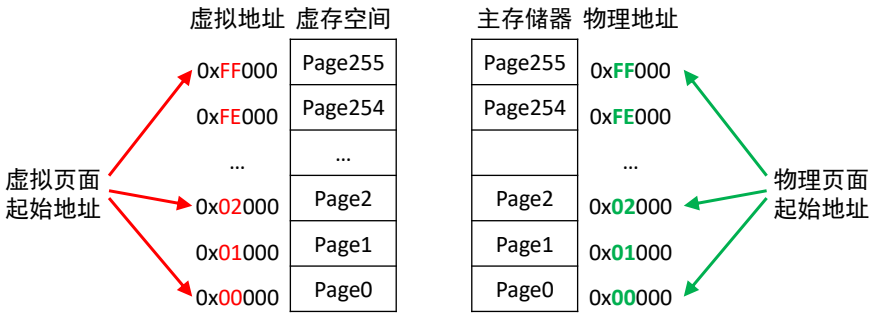
physical page #	offset

  - 物理页号(physical page number，PPN)，页内偏移(page offset)
    - PPN：PA去除offset后的高位部分
- 注意：由于虚地址空间与物理地址空间不必是一样大小，因此VPN与PPN也不必是一样大小的

计算机组成与实现

分页示例

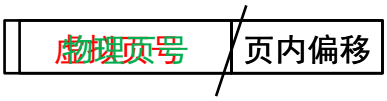
- 假设：页面容量为4KB；虚存空间、主存空间均为1MB
  - 页内偏移位数：$\log_2(4K) = 12$位
  - 虚拟页号位数：$\log_2(1M) - 12 = 20 - 12 = 8$位
  - 物理页号位数：$\log_2(1M) - 12 = 20 - 12 = 8$位



目录

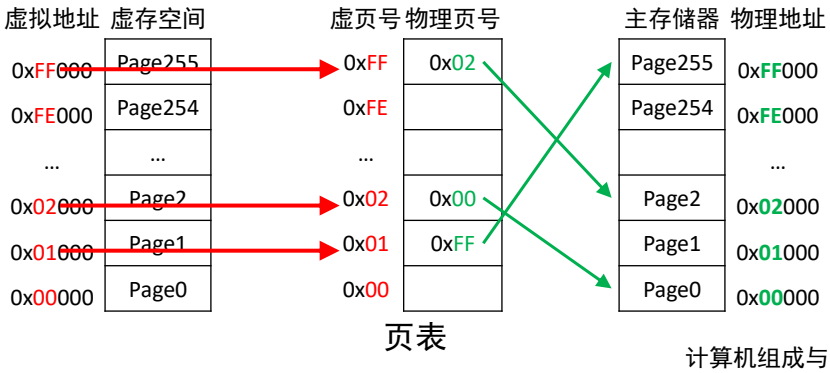
页表（page table，PT）

- 功能：输入虚拟地址（VA），输出对应的物理地址（PA）
  - 由于双方都是页面对齐的，因此VA的页内偏移=PA的页内偏移
  - 只需要根据虚拟页号得到相应的物理页号，然后再拼接页内偏移



页表（page table，PT）

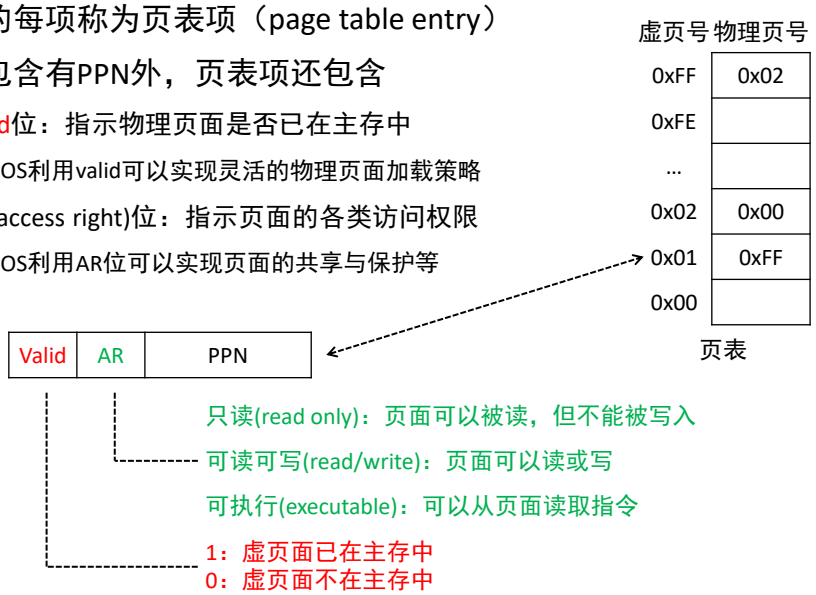
- 实现：数组
  - 虚拟页号（VPN）作为数组下标，数组单元中存储的是物理页号（PPN）



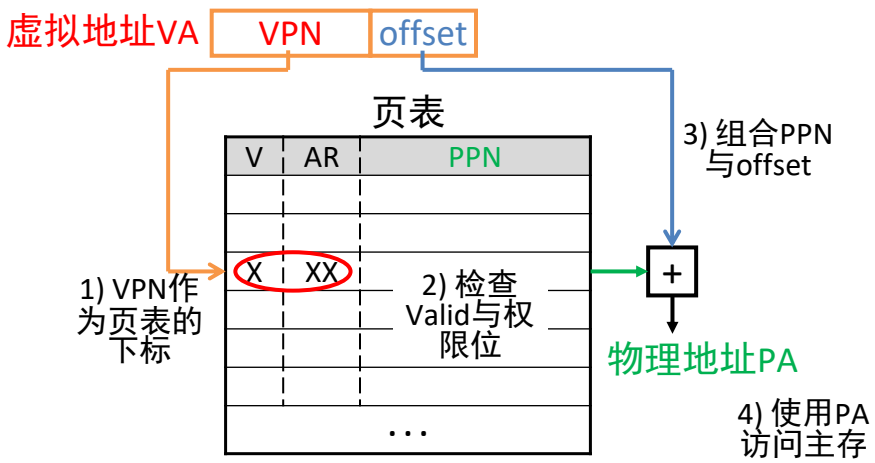
页表

页表的细节

- 页表的每项称为页表项（page table entry）
- 除了包含有PPN外，页表项还包含
  - Valid位：指示物理页面是否已在主存中
    - OS利用valid可以实现灵活的物理页面加载策略
  - AR(access right)位：指示页面的各类访问权限
    - OS利用AR位可以实现页面的共享与保护等



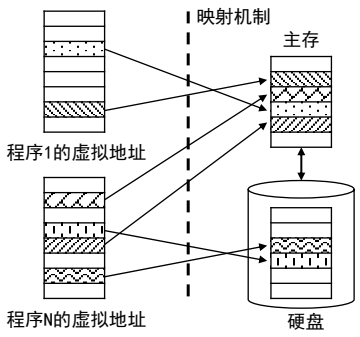
页表的细节



目录

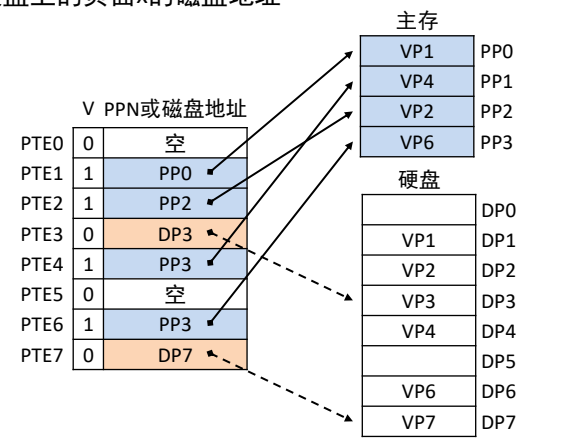
虚拟存储层次的基本内涵

- 在前述的分页基础上，可以模仿“cache-主存”层次结构，将硬盘引入，构成“主存-硬盘”的虚拟存储层次结构
- 优势1：给程序员提供一个空间巨大的存储视图
  - 程序员可用存储空间远大于主存，仅受硬盘容量限制
- 优势2：每个程序只使用它自己的虚拟地址，并且“认为”它是唯一运行的程序
  - 为程序开发(包括编译器)带来巨大便利
- 优势3：既在不同程序间实现了共享，又提供了必要的安全防护
  - 需要利用页面权限



虚存系统的重要概念：页表项扩展

- 由于引入了硬盘，因此一个虚页面可能存储在磁盘上，也可能已经装载到主存（某个物理页面）
- 因此，PTE不仅需要记录PPN，也需要记录相应的磁盘地址
  - 这里用DPx代表存储在硬盘上的页面x的磁盘地址

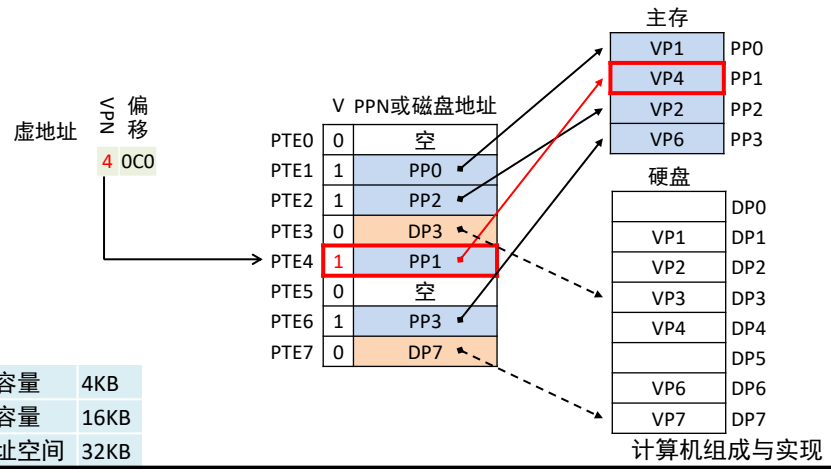


示例虚存

页面容量	4KB
主存容量	16KB
虚地址空间	32KB

虚存系统的重要概念：页命中

- 页命中(page hit)：cpu要访问的数据已经存储在主存中了
- 示例：cpu读40C0h（VPN为4，对应的页表项为PTE4）
  - PTE4的valid为1，说明VP4已经保存在主存中，其物理页号为PP1

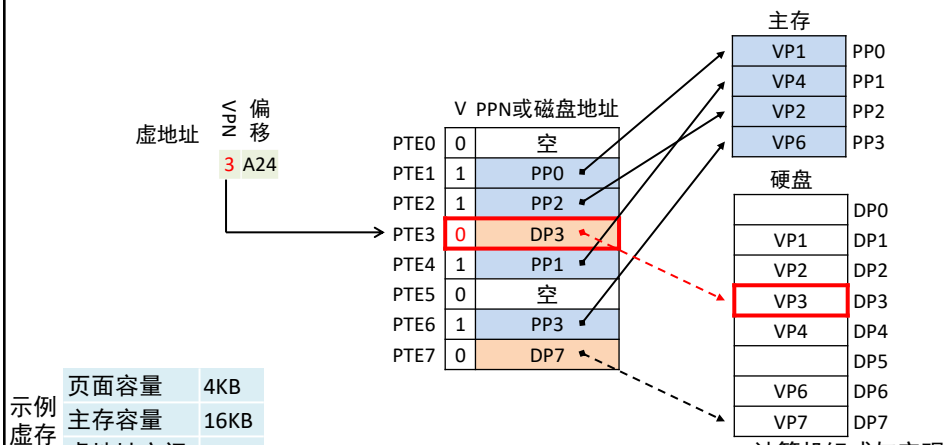


示例虚存

页面容量	4KB
主存容量	16KB
虚地址空间	32KB

虚存系统的重要概念：缺页

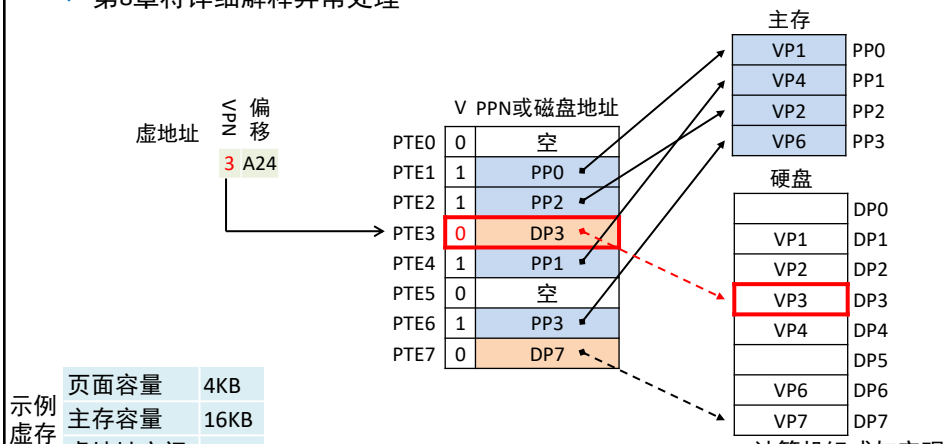
- 页命中(page miss)：cpu要访问的数据不在主存而在硬盘中
- 示例：lw指令读3A24h（VPN为3，对应的页表项为PTE3）
  - PTE3的valid为0，说明VP3不在主存而是在硬盘中，其硬盘地址为DP3



示例虚存		
	页面容量	4KB
	主存容量	16KB
	虚地址空间	32KB

虚存系统的重要概念：缺页处理1/5

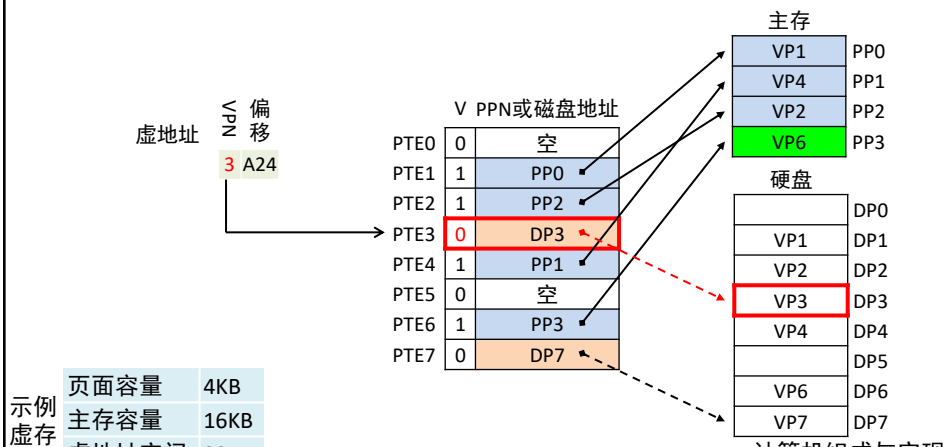
- 缺页会导致异常（exception ）
  - 异常发生后，cpu会跳转到异常处理程序（handler）
  - Cpu执行handler完成相应处理后再重新执行发生异常的指令（即lw）
  - 第8章将详细解释异常处理



示例虚存		
	页面容量	4KB
	主存容量	16KB
	虚地址空间	32KB

虚存系统的重要概念：缺页处理2/5

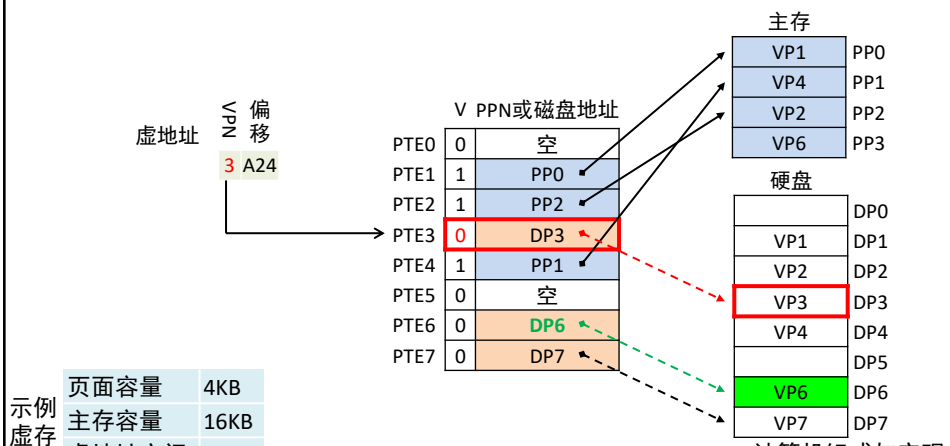
- 缺页会导致异常（exception ）
- 缺页异常handler：替换一个虚页（假设被替换的虚页是VP6）



示例虚存		
	页面容量	4KB
	主存容量	16KB
	虚地址空间	32KB

虚存系统的重要概念：缺页处理3/5

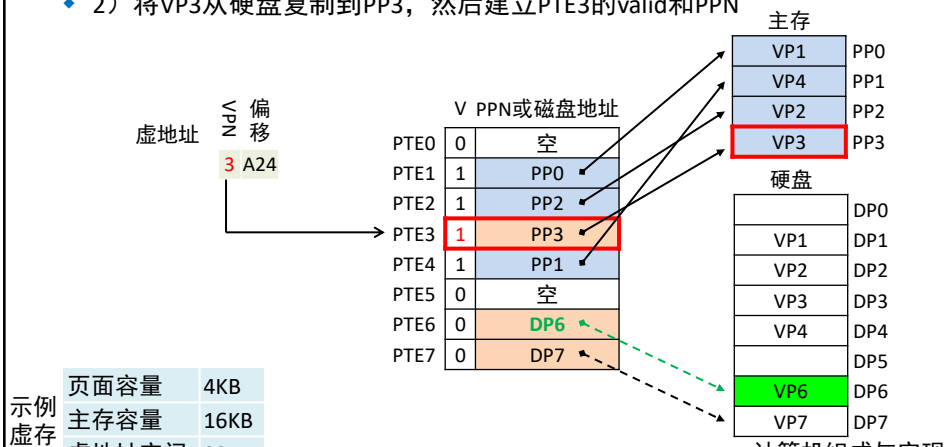
- 缺页会导致异常（exception ）
- 缺页异常handler：替换一个虚页（假设被替换的虚页是VP6）
  - 1）设置PTE6的valid为0，建立VP6的正确的硬盘地址



示例虚存		
	页面容量	4KB
	主存容量	16KB
	虚地址空间	32KB

虚存系统的重要概念：缺页处理4/5

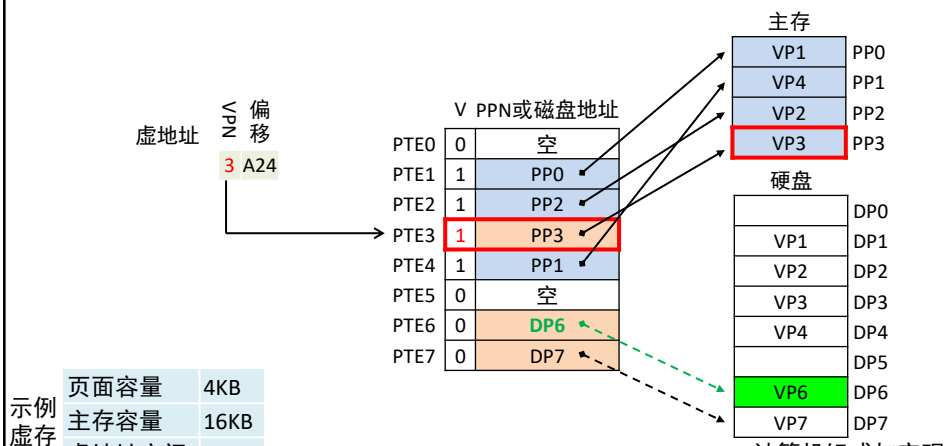
- 缺页会导致异常（exception ）
- 缺页异常handler：替换一个虚页（假设被替换的虚页是VP6）
  - 1）设置PTE6的valid为0，建立VP6的正确的硬盘地址
  - 2）将VP3从硬盘复制到PP3，然后建立PTE3的valid和PPN



示例虚存		
	页面容量	4KB
	主存容量	16KB
	虚地址空间	32KB

虚存系统的重要概念：缺页处理5/5

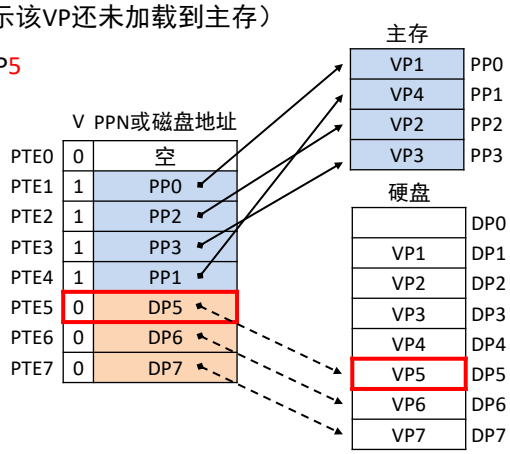
- 缺页异常handler返回后，cpu重新执行lw：页面命中！



示例虚存		
	页面容量	4KB
	主存容量	16KB
	虚地址空间	32KB

虚存系统的重要概念：分配页面

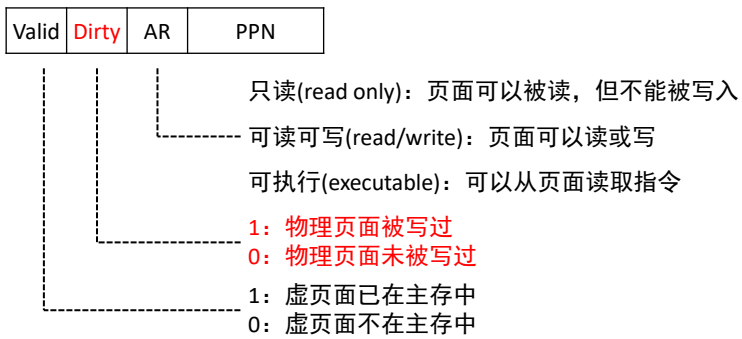
- 系统不可能完全预知程序的存储需求，因此需要动态分配虚页面
- 示例：C程序调用malloc()申请4KB内存
  - 假设分配的4KB对应VP5
  - 设置PTE5的valid为0（表示该VP还未加载到主存）
  - 设置PTE5的硬盘地址为DP5



示例虚存	
页面容量	4KB
主存容量	16KB
虚地址空间	32KB

虚存系统的重要概念：支持写

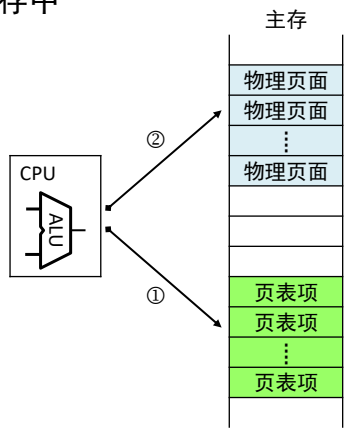
- 主存-硬盘：大约是6个数量级的性能差距
  - 由于性能差距过于悬殊，因此虚存系统必须采用write-back策略
- Dirty：用于指示页面是否被写过
  - OS在替换页面时，查看dirty就可以判断是否需要回写



目录

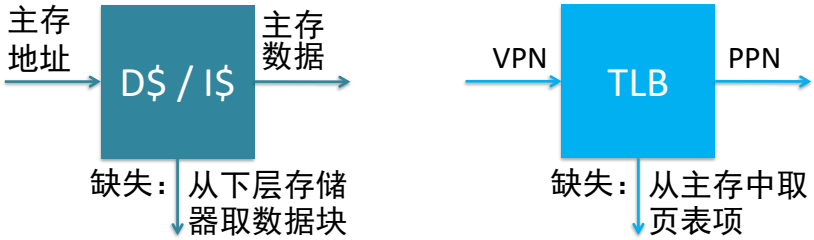
页表存储在哪里？

- 假设：32位虚地址，页面容量为4KB，页表项为4B
  - 1M个页面，每个页面的页表项4B，则总容量为4 x 2^{20} = 4MB
  - 对于4G主存，只有0.1%
- 页表对cache来说太大了，只能存储在主存中
- 这导致每个lw/sw需要2次主存访问！
  - ①读取PTE中的物理页号
  - ②根据物理地址访问主存单元
- 最好情况：PTE在cache中
  - 性能损失1~2个时钟周期
- 最坏情况：PTE不在cache中
  - 性能损失几十到上百个时钟周期



TLB（Translation Lookaside Buffer）

- 由于数据具有局部性特点，因此页表项也必然具有局部性
- TLB：专用于存储PTE的cache
  - TLB通常容量较小，典型的包含16 – 512个PTE
  - TLB性能与cache相当
  - TLB采用组相联结构（由于数量较少，通常相联度很高）



典型的TLB表项格式

- Valid、Dirty和AR：功能与前述的PTE中的一致
- Ref：用于实现LRU替换算法
  - 当页面被访问后，Ref被置位；OS周期性的清除该位
  - 如果Ref==1，则该页最近被访问过
- TLB Tag：VPN mod (TLB表项数)

Valid	Dirty	Ref	AR	TLB Tag	PPN
X	X	X	XX		

TLB示例

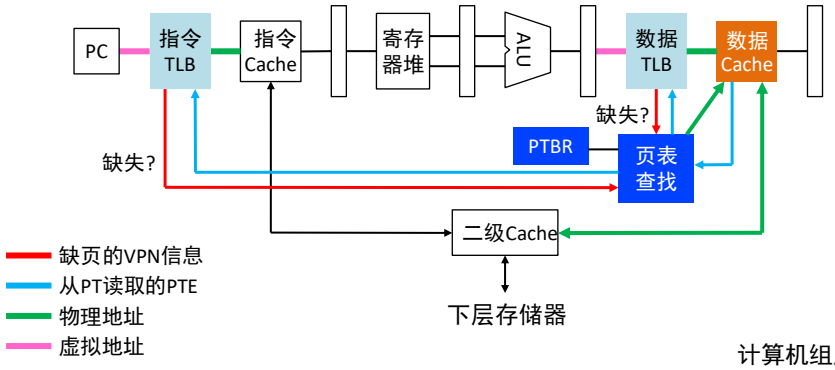
- 虚存系统参数
  - 16KB页面；40位虚地址；64GB主存；TLB采用2路组相联，共512表项
- TLB表项位数：45
  - Offset位数：$\log_2(16K) = 14$位
  - PPN位数：$\log_2(64G) - 14 = 36 - 14 = 22$位
  - VPN位数：$40 - 14 = 26$位
  - TLB组数：$512/2 = 256$组
  - Index位数：$\log_2(\text{TLB组数}) = \log_2(256) = 8$位
  - Tag位数：VPN位数 – Index位数 = 26 – 8 = 18位

Valid	Dirty	Ref	AR	TLB Tag	PPN
1	1	1	2	18	22

目录

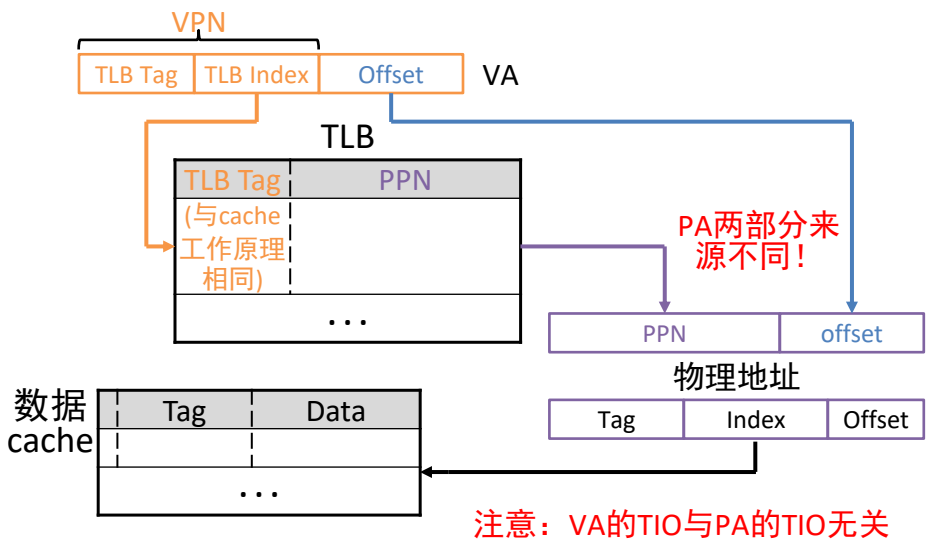
集成TLB的完整流水线

- 页表基地址寄存器（page table base register，PTBR）
  - 因为每个程序都有各自的页表，所以OS在调度某个程序运行时，会将该程序的页表在主存中的基地址加载至PTBR
- 页表查找
  - 当指令TLB和数据TLB发生缺失时，从存储在主存的页表中读取相应的PTE



计算机组成与实现

使用TLB完成虚实地址转换



计算机组成与实现

完整的虚实地址转换

- 1）检查TLB（输入：VPN，输出：PPN）
  - TLB命中：返回PPN给cache
  - TLB缺失：检查页表（在主存中）
    - 页表命中：加载PTE到TLB
    - 页表缺失：从硬盘中加载页面到主存，更新PTE，然后加载PTE到TLB
- 2）检查cache（输入：PA，输出：数据）
  - Cache命中：返回数据给cpu
  - Cache缺失：从主存中取数据，更新cache，然后返回数据给cpu

数据访问场景分析

- TLB缺失，页面缺失（可能）
  - 页面没有建立，则PTE必然没有，因此TLB缺失是合理的
- TLB命中，页表命中（不可能）
  - TLB命中就不会再去访问页表
- TLB缺失，Cache命中（可能）
  - TLB的PTE因访问其他页面被替换了，但之前的页面数据仍可在cache中
- 页表命中，Cache缺失（可能）
  - 页面已经建立并被访问过；但后续因cache替换导致数据不在cache中
- 页面缺失，Cache命中（不可能）
  - 页面缺失表明未建立页面，故主存中不可能有数据，cache也不可能命中

页面状态场景分析

- 假设：V=Valid，D=Dirty，R=Ref，分别与PTE和TLB的相应项一致
- 对于某一数据访问，分析场景合理性
  - Read，D = 1　不可能
  - Write，R = 1　合理
  - Read，V = 0　合理
  - Write，D = 0　不可能

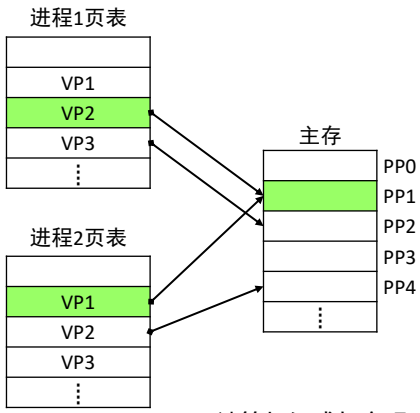
性能参数

- 虚拟存储系统同样采用命中率、缺失率、缺失代价
- TLB缺失率：TLB缺失次数占TLB总访问次数的比例
- 页表失效率：页表失效次数占页表总访问次数的比例

目录

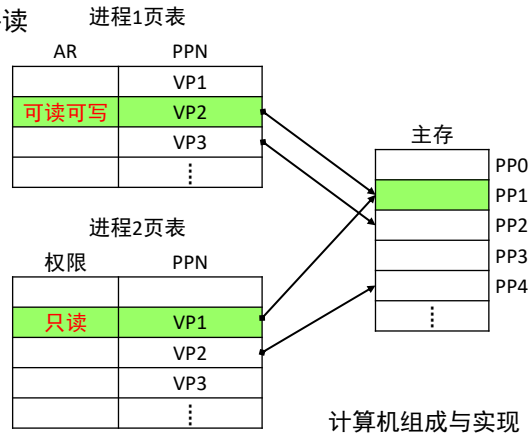
存储共享

- 多个程序在运行时，往往彼此之间需要共享数据
  - 例如：公共的库例程；传递信息
- 虚存系统可以很容易的支持数据共享
- 示例：进程1的VP2与进程2的VP1映射到同一个物理页面PP1
  - 这使得进程1（生产者）产生的数据，很容易就被进程2（消费者）获取



存储保护

- 出于安全考虑，计算机系统往往需要管控进程对存储器的读写
  - 例如：不允许进程修改只读数据，不允许进程修改OS内核代码和数据
- 示例：生产者-消费者模型
  - 进程1是生产者，因此可读可写
  - 进程2是消费者，因此只允许读



计算机组成与实现

目录

- CPU地址与存储器地址
- 基本原理
- 地址转换
- 页表
- 虚拟存储层次
- TLB
- 集成TLB与cache
- 存储共享与保护
- 页面替换
- 性能
- 小结

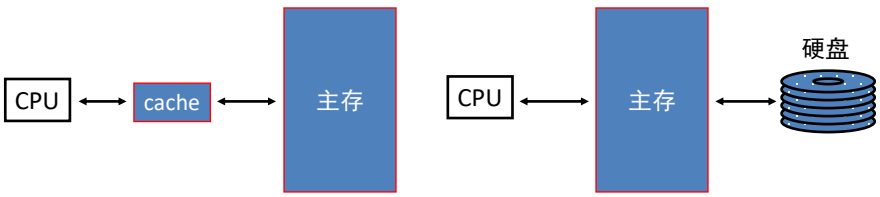
计算机组成与实现

页面替换

- 当程序要访问的页面不在主存时，会发生缺页，OS于是需要将该页面从硬盘中加载到主存中
- 假设主存中的物理页面都被占用了，那么就必须替换掉某个页面
- 如果被替换的页面很快就要被再次装入，则性能损伤会极大
  - 主存与硬盘的性能差距远远大于cache与主存的性能差距
- 与cache采用伪LRU算法不同，虚存系统通常会采用LRU算法

目录

典型的性能参数



Cache	请页
cache块 (≈32 bytes)	page (≈4KB)
cache miss rate (1% to 20%)	page缺失率 (<0.001%)
cache hit (≈1 cycle)	page命中 (≈100时钟周期)
cache缺失代价 (≈100时钟周期)	page缺失代价 (≈5M时钟周期)

换页对AMAT的影响1/2

- 参数
  - L1 cache HT = 1时钟周期，HR = 95%
  - L2 cache HT = 10时钟周期，HR = 60%（相对于L1的缺失而言）
  - DRAM = 200时钟周期（≈100ns）
  - Disk = 20,000,000时钟周期（≈10ms）
- AMAT（无换页）

  = 1 + 5%×10 + 5%×(1-60%)×200 = 5.5时钟周期
- AMAT（有换页）

  =5.5（$AMAT_{无换页}$）+ ?

换页对AMAT的影响2/2

- AMAT（有换页）

  = 5.5 + 5%×40%× (1-HR_{Mem})×20,000,000

  = 5.5 + 0.02× (1-HR_{Mem})×20,000,000

  - HR_{mem}：物理页面命中率
- 如果HR_{mem}为99%，AMAT =

  = 5.5 + 0.02×0.01×20,000,000 = 4005.5时钟周期
- 如果HR_{mem}为99.9%，AMAT =

  = 5.5 + 0.02×0.001×20,000,000 = 405.5时钟周期
- 如果HR_{mem}为99.9999%，AMAT =

  = 5.5 + 0.02×0.000001×20,000,000 = 5.9时钟周期

目录

小结

程序员看到的：
- 连续的主存
- 从特定地址启动
- 无限的主存
- 独占计算机

VM提供：
- 连续的存储视图
- 所有程序都从相同的地址启动
- 无限的主存
- 存储共享与保护

真实情况：
- 不连续的存储器
- 从主存任何地址启动
- 有限的主存
- 同时运行多个程序

VM实现：
- 把存储器分成块（页面）
- OS控制负责虚实转换的页表
- 主存是硬盘的cache
- TLB是页表的cache

计算机组成与实现